Supplementary material for GoT-WAVE: Temporal network alignment using graphlet-orbit transitions

D. Aparício[1], P. Ribeiro[1], T. Milenković[2] and F. Silva[1]

[1] CRACS & INESC-TEC, and the Department of Computer Science, Faculty of Sciences of the University of Porto, Porto, 4169-007, Portugal

[2] Department of Computer Science and Engineering, ECK Institute for Global Health, and Interdisciplinary Center for Network Science and Applications (iCeNSA), University of Notre Dame, Notre Dame, IN 46556, USA.

E-mail: daparicio@dcc.fc.up.pt

S1. Extracting Graphlet-orbit Transitions (GoTs)

Supplementary Algorithm S1 describes how we obtain exact counts of k-node GoTs of a given network. First, our method generates all k-node graphlets and their respective orbits and populates a g-trie with them (lines 2 and 3). A g-trie is a datastructure that efficiently stores input subgraphs and greatly speeds up their enumeration in a network [19]. The main idea of a g-trie is that subgraphs of a given size (say, 4-nodes) have common subtopologies. Thus, instead of searching each subgraph individually, g-tries obtain the exact occurrences of smaller subgraphs (say, 3-nodes) and grow them iteratively without starting computation from scratch. When they were first proposed, g-tries were a one or two orders of magnitude faster than state-of-the art algorithms, such as FANMOD [26] and Kavosh [10]. Other fast methods for subgraph enumeration include QuateXelero [11] and FASE [16]. More recently, analytic approaches such as PGD [20] and ESCAPE [17] have been proposed to greatly speed up graphlet enumeration. However, these latter approaches are not general (i.e., they can only be used for specific sets of graphlets, e.g., undirected graphlets of at most give nodes).

Note that our method generates the g-trie (for a given set of orbits) only once, stores it in a file and reuses it when necessary. The enumeration process is repeated for all T snapshots, storing the enumerated occurrences in a vector of the form of $\{Nodes_{\{1..k\}}, t, Orbits_{\{1..k\}}\}$. We use this representation so that, after we sort the vector (line 10), the occurrences of the same subgraph $Nodes_{\{1..k\}}$ are in subsequent positions of vector $Occs$ and sorted by t. We compare each consecutive pair of occurrences (Occ_1, Occ_2) and, if they belong to the same subgraph (i.e., the nodes are the same), we update the $GoTs$ of each node according to its orbit transition from t to $t + k$ (lines 11-14).

Consider the two possible 3-node undirected connected graphlets (chain and triangle) and their orbits from Supplementary Figure S3. A chain has two possible orbits, i.e., a node can be either at the center of the chain or in one of its leaves. A triangle has a single orbit, as all of its nodes are topologically equivalent. GoTs are the matrix of changes (transitions) between every possible pair of orbits across two consecutive snapshots. There are a total of $3 \times 3 = 9$ possible orbit transitions in Supplementary Figure S3. A node can remain in its previous orbit, be it a (A) chain-center, (D) chain-periphery or (I) triangle-node. Or, it can transition from the chain-center to the chain-periphery (B) or to a triangle-node (C), etc. All possibilities for the 3-node graphlets are illustrated in Figure 3; in practice, we use larger graphlets as well (see below). The matrix from Supplementary Figure S3 illustrates the GoTs of node x that we use as x's feature vector.

Algorithm S1 Obtain all k-node GoTs of network N with T snapshots

procedure ENUMERATEGOT(N, T, k)
  $\mathcal{O}_k$: generate all k-node orbits.
  GT_k = CREATEGTRIE($\mathcal{O}_k$)
  $GoTs : Nodes(N) \times |\mathcal{O}_k| \times |\mathcal{O}_k|$ tensor.

  for all $t \in \{1 : T\}$ **do**
    $\mathcal{S}_{N,t}$: snapshot t of N
    $Occs\{t\}$ = ENUMERATEORBITOCCURRENCES($\mathcal{S}_{N,t}, GT_k$)
  end for

  SORT($Occs$)
  for all $(Occ_1, Occ_2) \in Occs$ **do**
    if $Nodes(Occ_1) == Nodes(Occ_2)$ **then**
      for all $i \in \{1 : k\}$ **do**
        $GoTs[Node_i(Occ)][Orbit_i(Occ_1), Orbit_i(Occ_2)]$ ++
      end for
    end if
  end for
  return $GoTs$
end procedure

S2. Performance evaluation

In the following tests we measure potential improvement of GoT-WAVE over DynaWAVE as follows.

S2.1 On synthetic networks

Let us denote by S_G the (accuracy or running time) score of GoT-WAVE, and by S_D the score of DynaWAVE. Also, let us denote by G_A the relative gain of GoT-WAVE over DynaWAVE in terms of accuracy, and by G_T the relative gain GoT-WAVE over DynaWAVE in terms of running time. Since for accuracy, a larger score is better, we define $G_A = \frac{S_G - S_D}{min(S_G, S_D)} \times 100\%$. On the other hand, since for running time, a lower score is better, we define $G_T = \frac{S_D - S_G}{min(S_G, S_D)} \times 100\%$. In both cases, positive gain (i.e., a positive G_A or G_T value) would indicate improvement of GoT-WAVE compared to DynaWAVE, and negative gain (i.e., a negative G_A or G_T value) would indicate degradation of GoT-WAVE compared to DynaWAVE. For example, in terms of accuracy, if GoT-WAVE has accuracy of 1 and DynaWAVE has accuracy of 0.7, then $G_A = \frac{1-0.7}{0.7} \times 100\% = 43\%$ (i.e., GoT-WAVE is superior to DynaWAVE). On the other hand, if GoT-WAVE has accuracy of 0.7 and DynaWAVE has accuracy of 1, then $G_A = \frac{0.7-1}{0.7} \times 100\% = -43\%$ (i.e., GoT-WAVE is inferior to DynaWAVE). As another example, in terms of running time, if GoT-WAVE takes 2 seconds and DynaWAVE takes 6 seconds, then $G_T = \frac{6-2}{2} \times 100\% = 200\%$ (i.e., GoT-WAVE is superior to DynaWAVE). On the other hand, if GoT-WAVE takes 6 seconds and DynaWAVE takes 2 seconds, then $G_T = \frac{2-6}{2} \times 100\% = -200\%$ (i.e., GoT-WAVE is inferior to DynaWAVE).

S2.2 On real networks

We denote the objective scores of the ideal and method-produced alignments for noise n by $S_{i,n}$ and $S_{p,n}$, respectively. The expectation is that a good method's produced objective score should be similar to the method's ideal objective score, i.e., $|S_{p,n} - S_{i,n}|$ should be as close as possible to 0. Also, since we want to account for scaling (e.g., the difference of 0.1 between 0.9 and 0.8 is not the same as the difference of 0.1 between 0.3 and 0.2), we divide the difference between the produced and ideal alignment by their maximum, i.e., $max(S_{p,n}, S_{i,n})$. With these points is mind, we compute the distance $dis(S_p, S_i)$ over all considered noise levels n (from 0% to 20%) as: $dis(S_p, S_i) = \sum_{n=0\%}^{20\%} \frac{|S_{p,n} - S_{i,n}|}{max(S_{p,n}, S_{i,n})}$

For each real network, we compute this distance for each of GoT-WAVE and DynaWAVE. Then, we summarize gain of GoT-WAVE compared to DynaWAVE as follows. Let us denote by S_G the distance score of GoT-WAVE, and by S_D the score of DynaWAVE. Since a lower distance score is better, we compute the relative gain of GoT-WAVE over DynaWAVE, denoted by G_O, as: $G_O = \frac{S_D - S_G}{min(S_G, S_D)} \times 100\%$. Positive gains mean than GoT-WAVE is superior to DynaWAVE and negative gains mean that GoT-WAVE is inferior to DynaWAVE.

Second, we compare GoT-WAVE and DynaWAVE in terms of node correctness (see above). Let us denote by S_G the node correctness of GoT-WAVE, and by S_D the node correctness of DynaWAVE. Since higher node correctness is better, we compute the relative gain of GoT-WAVE over DynaWAVE, denoted by G_{NC}, as: $G_{NC} = \frac{S_G - S_D}{min(S_G, S_D)} \times 100\%$. Again, positive gains mean than GoT-WAVE is superior to DynaWAVE and negative gains mean that GoT-WAVE is inferior to DynaWAVE.

S3. Temporal random graph models

We generate networks with 24 snapshots each. In each snapshot, new nodes and edges are added to it, growing the network but keeping edge density. Node arrival is either linear or exponential. We set the arrival function of each model based on similar models [12]. Which new edges are added is specific to each model (detailed below). Edge density is set at ≈1% for all models, mimicking real-world networks (such as PINs [14]), and remains stable for all snapshots (e.g., this stability was observed in social networks [7]). Each network starts with 100 nodes and grows to 1,000 nodes. We generate ten networks for each of the five models, for a total 50 networks with 24 snapshots each, or 1200 snapshots.

Next, we give specific details on the models. Supplementary Table S1 summarizes the models.

- **Erdős-Rényi (Random) [3]:** Two nodes are chosen at random and connected. Past edges are kept.
- **Barabási-Albert (ScaleFree) [2]:** Two nodes u and v are chosen at random but they become connected only if $\frac{max(deg(u),deg(v))}{|E|} > r$, where r is a randomly generated value $\in [0, 1]$. Therefore, nodes with higher degrees have a bigger chance of gaining new connections ("the rich get richer"). Past edges are kept.
- **Watts–Strogatz (Small-world) [25]:** At $t = 0$ an initial ring network is created, where each node is connected to $\approx k$ neighbors. Since $N_0 = 100$ and the edge density is 1%, $k = 1$ for $t = 0$. Edges are then randomly rewired with probability $\beta = 0.2$. For $t > 0$, new nodes arrive and a ring is formed again (with a possibly different k) while keeping the rewired connections previously added. Rewiring is again performed, both on the new ring and on the old randomized edges. Therefore, past edges might disappear.
- **Geometric gene duplication (Geo-GD) [18]:** Initially k seed-nodes are placed close-by in a two-dimensional space: $d(u, v)^2 < \epsilon$. As suggested by [18], $\epsilon = 5 \cdot 10^{-2}$ and $k = 5$ are used. Nodes are incrementally added to the network one at a time, and each new node u is placed at a distance $d(u, f)^2 \leq \epsilon$ from a random *father-node* f already in the network. The model includes parameter p which controls how likely node u is to *cut-off* from f; for our purposes $p = 0.2$ since it gives origin to realistic PPI networks [18]. When a node cuts-off from its father, it is placed at a distance of 10ϵ at most. Node additions stop when the desired number of nodes is achieved and the closest 1% edges are added to snapshot t, while the other possible edges remain unactivated. It is possible that past edges disappear since close edges in snapshot t are not guaranteed to remain in the 1% closest edges of snapshot $t + 1$.
- **Scale-free gene duplication (ScaleFree-GD) [22]:** After a few seed edges are added to the network, each new node is (i) connected to a random father-node and it (ii) copies the father's connections. The model has two parameters: p controls the likelihood of child- and father-nodes being connected by an edge, and q sets the probability of the child-node keeping his father's connections to other nodes. For our purposes, $p = 0.3$ and $q = 0.7$ since these values generated realistic PPI networks [18]. The model also sets a 50% chance that when an edge is not successfully replicated from father to child either (a) the father keeps the edge but the child does not copy it or (b) the child steals the connection from the father; therefore, past edges can be lost.

S4. Computing ROC curves

With each of GoT-WAVE and DynaWAVE, we align all pairs of synthetic networks. We compute objective function scores of all alignments. Objective scores vary from 0 (i.e., the networks are completely different) to 1 (i.e., the networks are exactly alike). We consider an objective score threshold of k, also varying from 0 to 1. Then, a pair of networks is: i) a true positive if their alignment's objective score is k or higher and the networks belong to the same model, ii) a false positive if their alignments objective score is k or higher but the networks belong to different models, iii) a false negative if their alignment's objective score is lower than k but the networks belong to the same model, and iv) a true negative if their alignment's objective score is lower than k and the networks belong to different models.

More formally, if G are H are two networks being aligned, $c(X)$ is the class (i.e., model) of the network and $Score(G, H)$ is their objective score, then, for a given threshold k:

$$Label(G, H, k) = \begin{cases} \text{True Positive} & Score(G, H) \geq k, c(G) = c(H) \\ \text{False Positive} & Score(G, H) \geq k, c(G) \neq c(H) \\ \text{True Negative} & Score(G, H) < k, c(G) \neq c(H) \\ \text{False Negative} & Score(G, H) < k, c(G) = c(H) \end{cases}$$

Then, for each level k, we compute the true positive rate (TPR)) and the false negative rate (FPR).

$$TPR(k) = \frac{|\text{True Positives}|}{|\text{True Positives} \cup \text{False Negatives}|} \tag{TPR}$$

$$FPR(k) = \frac{|\text{False Positives}|}{|\text{False Positives} \cup \text{True Negatives}|} \tag{FPR}$$

We vary k in increments of 0.01, resulting in 1000 pairs of TPR versus FPR. We then compute the area under the receiver operating characteristic (AUROC) curve as follows:

$$AUROC = \sum_{k=0.01}^{1} TPR(k)\Delta FPR(k) \tag{1}$$

S5. Randomization schemes

We insert noise using three different randomization schemes. For undirected networks, we use an established randomization scheme [6], which we refer to as *undirected randomization*. This scheme chooses two random events and swaps their timestamps with some probability. For directed networks, we use a variation of the above scheme that also randomly swaps edge directions with some probability, which we refer to as *directed randomization*. For directed networks, we use an additional scheme that only swaps the edge direction of events but not their timestamps, which we refer to as *pure directed randomization*.

We study 10 noise levels, from 0% to 20% in increments of 2%. We produce five random networks for each noise level. First, for a given method, at each noise level, for each alignment, we compute its objective function score. Ideally, the score decreases as the network is aligned to progressively noisier versions. Additionally, since we know the perfect alignment between the original network and its randomized versions (as their nodes are the same), we compute the ideal objective score, i.e., the quality of the perfect alignment, as measured by DynaWAVE's and GoT-WAVE's objective function. We then measure the distance to the each method's ideal alignments (details in Supplementary Section 2.2).

Next, we provide more details on each scheme.

S5.1 Undirected randomization

This randomization scheme was proposed in [6] and used in [23, 24]. Given the original undirected dynamic network $\mathcal{S}_N$ a randomization percentage p, one randomly picks edge e_1 to be rewired. We then pick another random edge e_2 and, with probability p, we rewire the two events. That is, given $e_1 = (u, v, S_i, S_f)$ and $e_2 = (u', v', S'_i, S'_f)$ (where S_i and S'_i are the starting snapshots, and S_f and S'_f are the ending snapshots) we do one of the following transformations with 50% probability:

- $e_1 = (u, v, S_i, S_f) \rightarrow e_1 = (u, v', S_i, S_f)$ and
  $e_2 = (u', v', S'_i, S'_f) \rightarrow e_2 = (u', v, S_i, S_f)$, or
- $e_1 = (u, v, S_i, S_f) \rightarrow e_1 = (u, u', S_i, S_f)$ and
  $e_2 = (u', v', S'_i, S'_f) \rightarrow e_2 = (v, v', S_i, S_f)$, or

If the transformation was performed, e_1 and e_2 are both taken out of the list of edges to be rewired. Otherwise, only e_1 is taken out. The process is followed until no edges are to be rewired.

S5.2 Directed randomization

This randomization scheme is adapted from [6] to directed networks. Given the original directed dynamic network $\mathcal{S}_N$ a randomization percentage p, one randomly picks edge e_1 to be rewired. We then pick another random edge e_2 and, with probability p, we rewire the two events. That is, given $e_1 = (u, v, S_i, S_f)$ and $e_2 = (u', v', S'_i, S'_f)$ (where S_i and S'_i are the starting snapshots, and S_f and S'_f are the ending snapshots) we do one of the following transformations with 50% probability:

- $e_1 = (u, v, S_i, S_f) \rightarrow e_1 = (u, v', S_i, S_f)$ and
  $e_2 = (u', v', S'_i, S'_f) \rightarrow e_2 = (u', v, S_i, S_f)$, or
- $e_1 = (u, v, S_i, S_f) \rightarrow e_1 = (u, u', S_i, S_f)$ and
  $e_2 = (u', v', S'_i, S'_f) \rightarrow e_2 = (v, v', S_i, S_f)$, or

If the transformation was performed, an additional parameter γ controls edge reversal. So, with probability γ, one performs the transformation for each edge:

- $e_k = (x, y, S_n, S_m) \rightarrow e_k = (y, x, S_n, S_m)$

Then, e_1 and e_2 are both taken out of the list of edges to be rewired. Otherwise, only e_1 is taken out. The process is followed until no edges are to be rewired.

S5.3 Pure directed randomization

Given the original dynamic directed network $\mathcal{S}_N$, one randomly picks edge e_1 to be rewired. That is, given $e_1 = (u, v, S_i, S_f)$ (where S_i is the starting snapshot, and S_f is the ending snapshot), and a randomization percentage p, we the following transformations with p% probability:

- $e_1 = (u, v, S_i, S_f) \rightarrow e_1 = (v, u, S_i, S_f)$

Otherwise, e_1 is kept as it was. e_1 is taken out of the list of edges to be rewired and the process is followed until no edges are to be rewired.

Supplementary Tables

Table S1: Set of graph models used in our experiments. All networks (regardless of the model) have 24 snapshots, start with 100 nodes, grow until they reach 1000 nodes, and have edge density of = 1% in all snapshots. Node arrival (linear or exponential) is set to what was reported in [12] for similar models. How nodes are connected (i.e., how new edges are added) depends on the model (see Supplementary Section S3).

Model	Node arrival	New edges
Random	linear	random
ScaleFree	exponential	preferential Attachment
Small-world	linear	ring + rewire ($\beta = 0.2$)
Geo-GD	linear	duplication w/ cut-off ($p = 0.2$)
ScaleFree-GD	exponential	duplication w/ edge loss ($p = 0.3$, $q = 0.7$)

Table S2: Results on synthetic networks for $\alpha = 0$ and $\alpha = \frac{1}{2}$. In parentheses, we show gain in performance of GoT-WAVE compared to DynaWAVE.

	AUROC	
α	DynaWAVE	GoT-WAVE
0	0.59	**0.78 (+32%)**
$\frac{1}{2}$	0.54	0.70 (+30%)

(a) Accuracy.

Model	DGDVs	GoTs
Random	26s	**22s (+18%)**
ScaleFree	**22s**	25s (-14%)
Small-world	23s	**4s (+475%)**
Geo-GD	34s	**11s (+210%)**
ScaleFree-GD	16s	**12s (+33%)**
Total	121s	**74s (+64%)**

(b) Feature extraction times.

Table S3: Average time to align two networks when using DGDVs or GoTs. We compute the time of aligning each model (e.g., the time to align each of the **Random** networks with any other networks). Since each of the 5 models m has 10 instances n (i.e., networks), we consider $\frac{(n-1)\times n}{2} = 45$ alignments of a given network to networks of its own model (e.g., align two **Random** networks) and $n \times (m-1) \times n = 400$ alignments to networks of different models (e.g., align a **Random** network with a **ScaleFree** network). For each model, we average the time over all 445 considered alignments.

Model	DGDVs	GoTs
Random	6.219s	**6.078s (+2%)**
ScaleFree	6.196s	**6.056s (+2%)**
Small-world	6.181s	**6.049s (+2%)**
Geo-GD	6.027s	**5.925s (+2%)**
ScaleFree-GD	6.009s	**5.868s (+2%)**
Total	30.632s	**29.976s (+2%)**

Table S4: Real temporal networks used in our experiments.

Network	Nodes	Events	Snaps.	Description
zebra [21]	27	500	57	Zebra proximity network
yeast [23]	1,004	10,403	8	Yeast PIN
aging [4]	6,300	76,666	38	Human aging PIN
school [5]	327	7,388	5	School proximity network
gallery [9]	420	22,476	16	Gallery proximity network
arxiv [13]	2,504	138,495	7	Paper co-authorships
emails [15]	167	8,771	9	E-mail communication
tennis [1]	876	103,938	42	Player dominance network

Table S5: Node correctness when aligning an undirected real network to itself (noise = 0). In parentheses, we show relative improvement (positive gain) or degradation (negative gain) in performance of GoT-WAVE compared to DynaWAVE. In bold, we show the best result for each network.

	(a) $\alpha = 0$		(b) $\alpha = \frac{1}{2}$	
Network	DynaWAVE	GoT-WAVE	DynaWAVE	GoT-WAVE
zebra	**0.926 ± 0.05**	0.578 ± 0.09 (-60%)	0.911 ± 0.04	0.615 ± 0.14 (-48%)
yeast	**0.966 ± 0.01**	0.924 ± 0.01 (-5%)	**0.966 ± 0.01**	0.919 ± 0.01 (-5%)
aging	0.912 ± 0.01	0.942 ± 0.01 (+3%)	**0.959 ± 0.01**	0.955 ± 0.01 (-0.4%)
arxiv	0.340 ± 0.02	0.446 ± 0.02 (+31%)	**0.658 ± 0.01**	0.602 ± 0.04 (-9%)
gallery	0.507 ± 0.03	0.485 ± 0.03 (-5%)	**0.557 ± 0.01**	0.531 ± 0.01 (-5%)
school	0.735 ± 0.03	0.861 ± 0.03 (+17%)	**0.973 ± 0.01**	0.971 +- 0.01 (-0.2%)

Table S6: Results on undirected real networks in terms of (a) feature extraction times and (b) number of subgraph occurrences (i.e., number of dynamic graphlets or GoTs found on the network). GoTs induce many more occurrences than DGDVs, and this is especially true for denser networks, such as **aging**, **arxiv** and **school**. Due to our fast enumeration algorithm based on g-tries [19], GoTs' extraction is still faster for the sparser networks, namely **zebra**, **yeast** and **school**. In parentheses, we show relative improvement (positive gain) or degradation (negative gain) in performance of GoT-WAVE compared to DynaWAVE. We assume that more occurrences means degradation. Thus, GoT-WAVE shows a much higher degradation in terms of number of occurrences than in execution time (-1,886% versus -142%), showcasing our enumeration algorithm's efficiency.

	(a)		(b)	
Network	Execution time		#Occurrences	
	DGDVs	GoTs	DGDVs	GoTs
zebra	0.06s	**0.02s (+200%)**	9.676×10^3	$\mathbf{7.792 \times 10^3}$ **(+24%)**
yeast	86s	**65s (+32%)**	$\mathbf{7.160 \times 10^6}$	5.815×10^7 (-712%)
aging	**202s**	696s (-245%)	$\mathbf{1.532 \times 10^7}$	5.510×10^8 (-3,496%)
school	4s	**2s (+100%)**	$\mathbf{3.765 \times 10^5}$	2.352×10^6 (-525%)
arxiv	**586s**	1,360s (-132%)	$\mathbf{6.178 \times 10^7}$	1.067×10^9 (-1,627%)
gallery	**6s**	14s (-133%)	$\mathbf{5.864 \times 10^5}$	1.432×10^7 (-2,341%)
Total	**884s**	2,137s (-142%)	$\mathbf{8.523 \times 10^7}$	1.693×10^9 (-1,886%)

Table S7: Average time to align two networks when using DGDVs or GoTs. We compute the time of aligning a network with each of its randomized versions. Since each network randomization has ten noise levels, each comprised of five networks, we consider a total of 50 alignments per network, and average out the times.

Network	DGDVs	GoTs
zebra	**4.50s**	5.36s (-19%)
yeast	**55.32s**	82.46s (-49%)
aging	2,264.92s	**1,209.58s (+87%)**
arxiv	416.40s	**249.98s (+67%)**
gallery	20.36s	**19.70s (+3%)**
school	20.46s	**14.72s (+39%)**
Total	2,781.95s	**1,586.79(75%)**

Table S8: Node correctness when aligning a network to itself ($\alpha = 0$). In parentheses, we show gain in performance of GoT-WAVE over DynaWAVE.

Network	DynaWAVE	GoT-WAVE		
	Undirected-4	Undirected-4	Directed-3	Directed-4
emails	**0.85 ± 0.02**	0.81 ± 0.03	0.83 ± 0.01 (-2%)	0.81 ± 0.02
tennis	0.74 ± 0.01	0.84 ± 0.03	**0.85 ± 0.02 (+15%)**	0.81 ± 0.02

Table S9: Results on directed real networks in terms of feature extraction times when using DGDVs or GoTs. For DGDVs, we extract dynamic graphlets with up to four nodes and up to six events, as suggested in [8]. For GoTs, we extract undirected GoTs with up to four nodes, directed GoTs with up to three nodes, and directed GoTs with up to four nodes. In parentheses, we show relative improvement (positive gain) or degradation (negative gain) in performance of GoT-WAVE compared to DynaWAVE. In bold, we show the best result for each network.

Network	DGDVs	GoTs		
	Undirected-4	Undirected-4	Directed-3	Directed-4
tennis	29.09s	113.07s (-289%)	**2.90s (+903%)**	128.43s (-341%)
emails	5.19s	5.65s (-9%)	**0.21s (+2,371%)**	7.84s (-51%)
Total	34.28s	118.72s (-246%)	**3.11s (+1,002%)**	136.27s (-297%)

Supplementary Figures

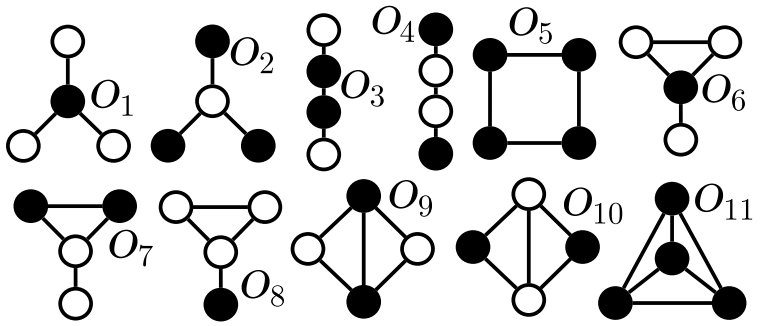


Figure S1: Graphlets are small non-isomorphic subgraphs. Different node positions (or symmetry groups) in a graphlet are called orbits. This figure shows all 11 orbits of all six undirected 4-node graphlets. Nodes that are in the same orbit (i.e., are topologically equivalent) in a given graphlet are colored in black. For example, o_1 and o_2 are two possible orbits of a 4-node star, orbits o_3 and o_4 are two possible orbits of a 4-node chain, etc. Graphlets are a general concept (e.g., not specific to a given size, and edge direction can be incorporated).

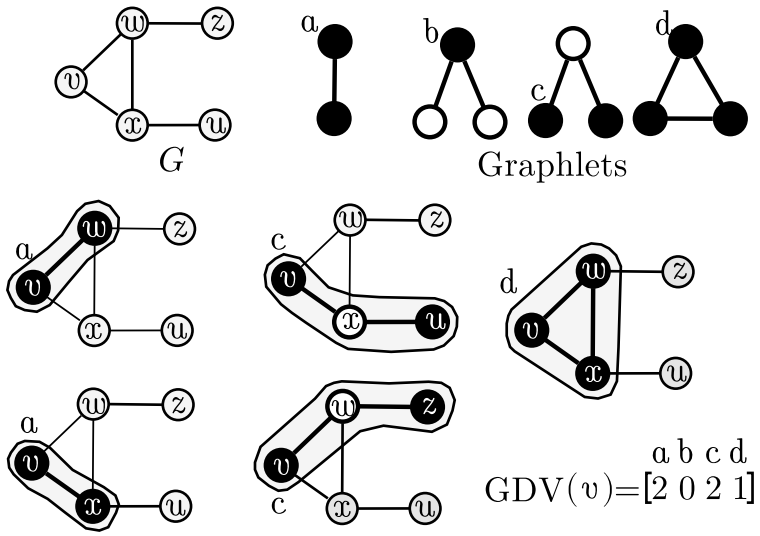


Figure S2: Illustration of GDV(v) that counts how many times v participates in each of the orbits a, b, c and d of all undirected 2-3-node graphlets. In this example, v touches orbit a twice (i.e., has degree of two), the periphery of a 3-node chain (orbit c) twice, and a triangle (orbit d) once.

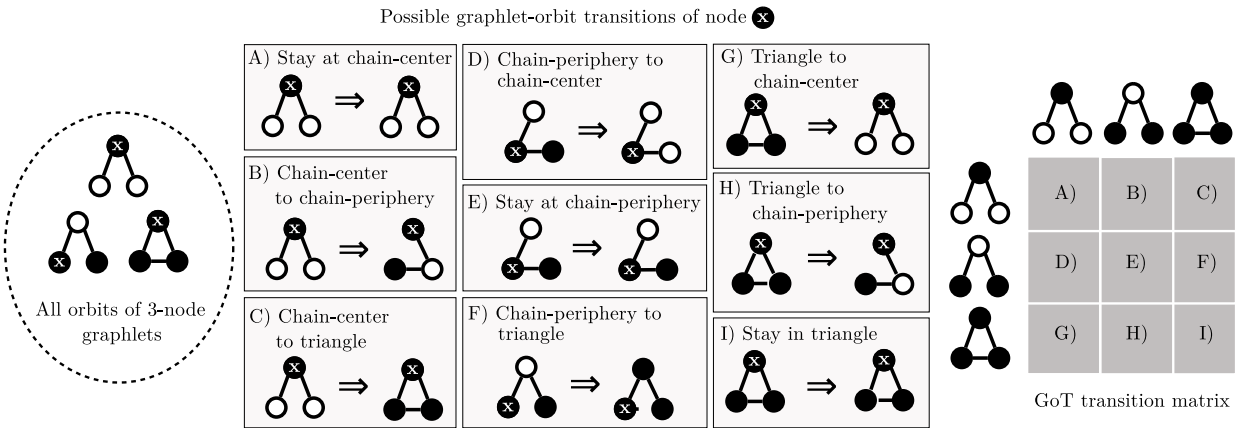
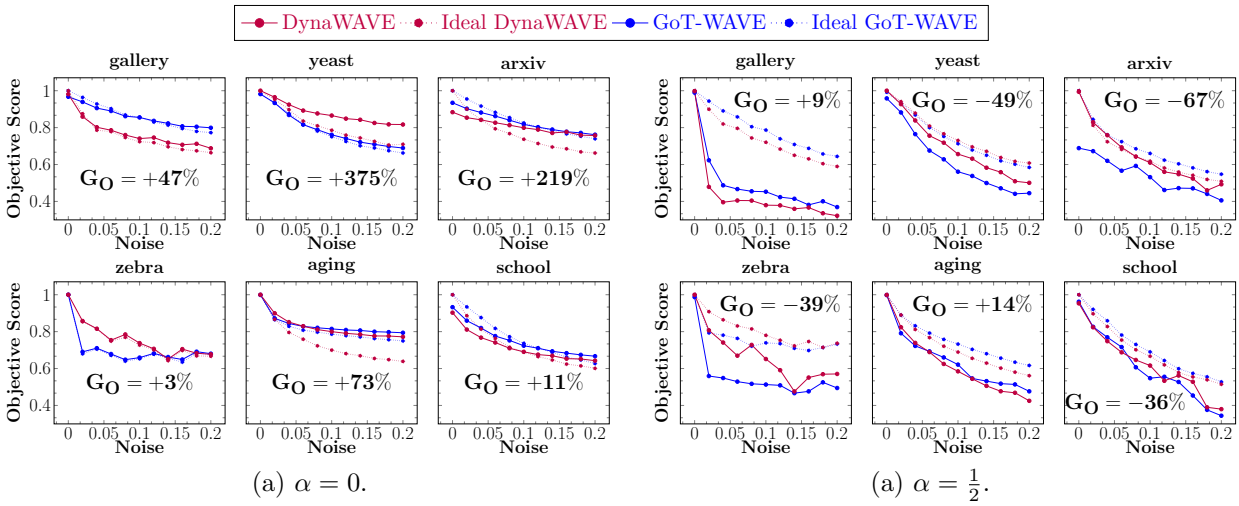


Figure S3: All possible graphlet-orbit transitions (GoTs) of 3-node undirected graphlets and the corresponding GoT matrix. Node x is the node being considered (whose GoT matrix is shown), and black nodes are in the same orbit as x. Each cell (i, j) of the GoT matrix represents the number of times node x transitions from orbit i to orbit j.



(a) $\alpha = 0$. (a) $\alpha = \frac{1}{2}$.

Figure S4: Comparison between GoT-WAVE and DynaWAVE on undirected networks in terms of how well their alignments' objective scores match the objective scores of ideal alignments, when (a) only node conservation is optimized ($\alpha = 0$) and (b) both node and edge conservation are optimized ($\alpha = \frac{1}{2}$). Recall that G_O is the relative gain of GoT-WAVE over DynaWAVE (positive: GoT-WAVE is superior; negative: DynaWAVE is superior).

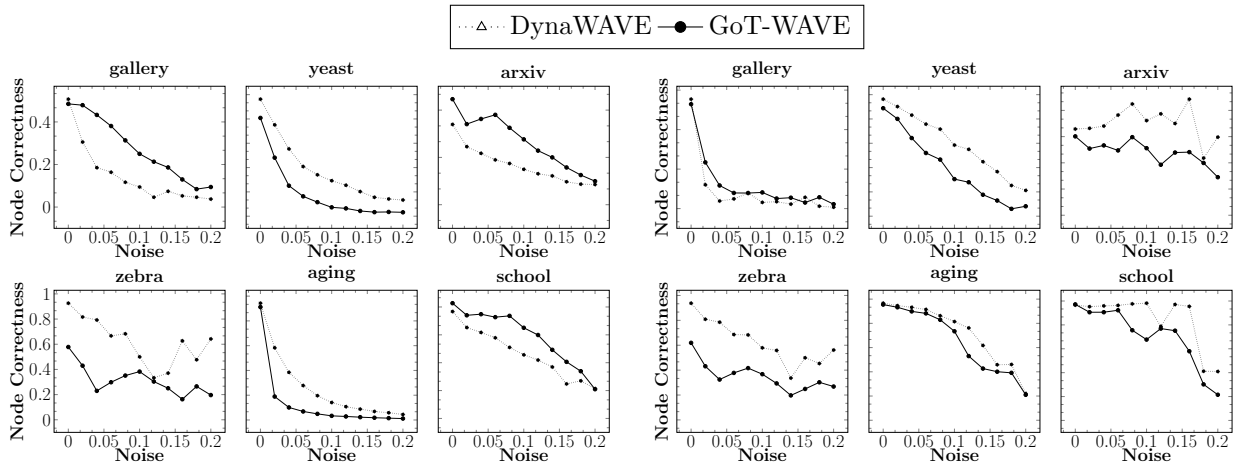


Figure S5: Comparison between GoT-WAVE and DynaWAVE on undirected networks in terms of node correctness, when (a) only node conservation is optimized ($\alpha = 0$) and (b) both node and edge conservation are optimized ($\alpha = \frac{1}{2}$). The higher the node correctness, the better the method.

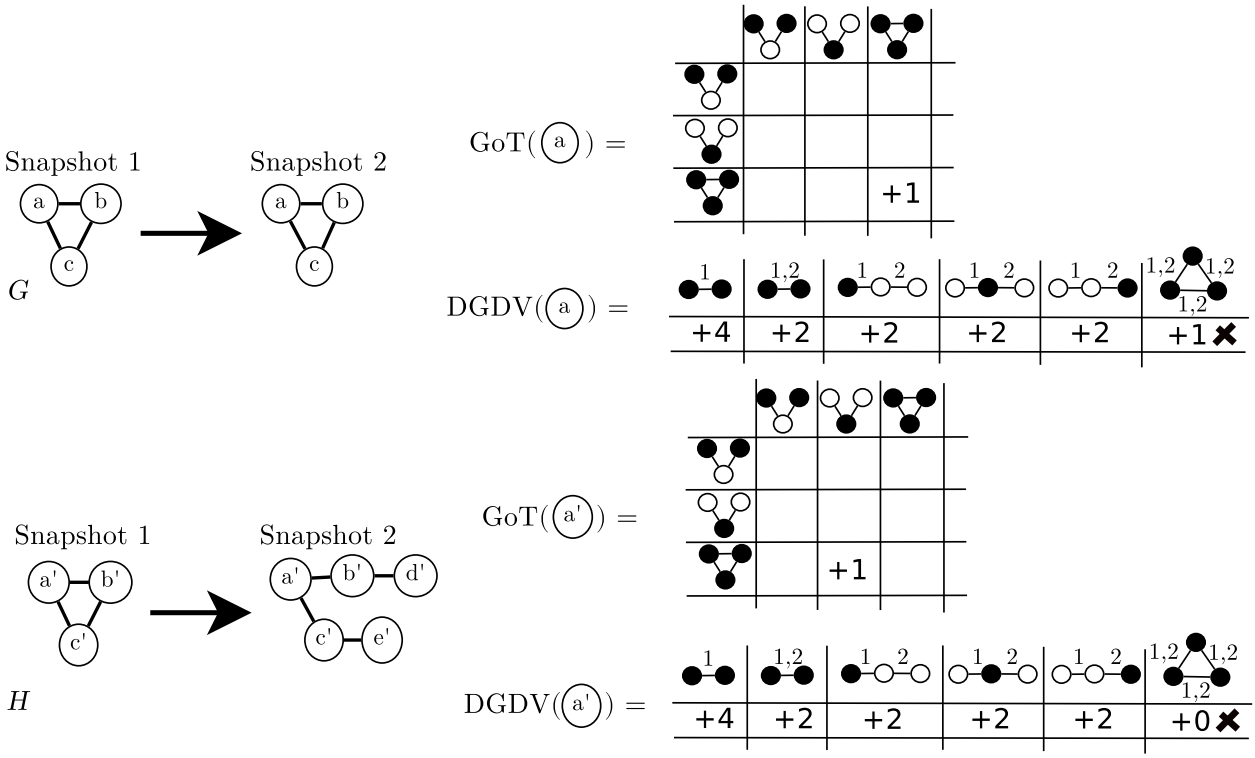


Figure S6: In the main paper we observe that, when both node and edge conservation are considered ($\alpha = \frac{1}{2}$), in contrast to when only node conservation is considered ($\alpha = 0$), DynaWAVE has better results than GoT-WAVE consistently. Here we hypothesize why that might the case with an example. When we use GoTs, node a from network G and node a' from network H are correctly identified as different (i.e., node a and node a' have different GoTs). When we use DGDVs, node a and node a' are incorrectly identified as similar/equal (i.e., node a and node a' have the same DGDVs). For details on DGDVs we refer the reader to [8]. DGDVs can not distinguish between nodes a and a' because a dynamic graphlet only allows for one event per time-step. For instance, the last graphlet in the DGDVs is not allowed (and not calculated). Thus, for this case, DGDVs can only distinguish nodes a and a' when edge conservation is also considered. Cases such as these show why DGDVs have bigger benefits in using DWEC than GoTs.

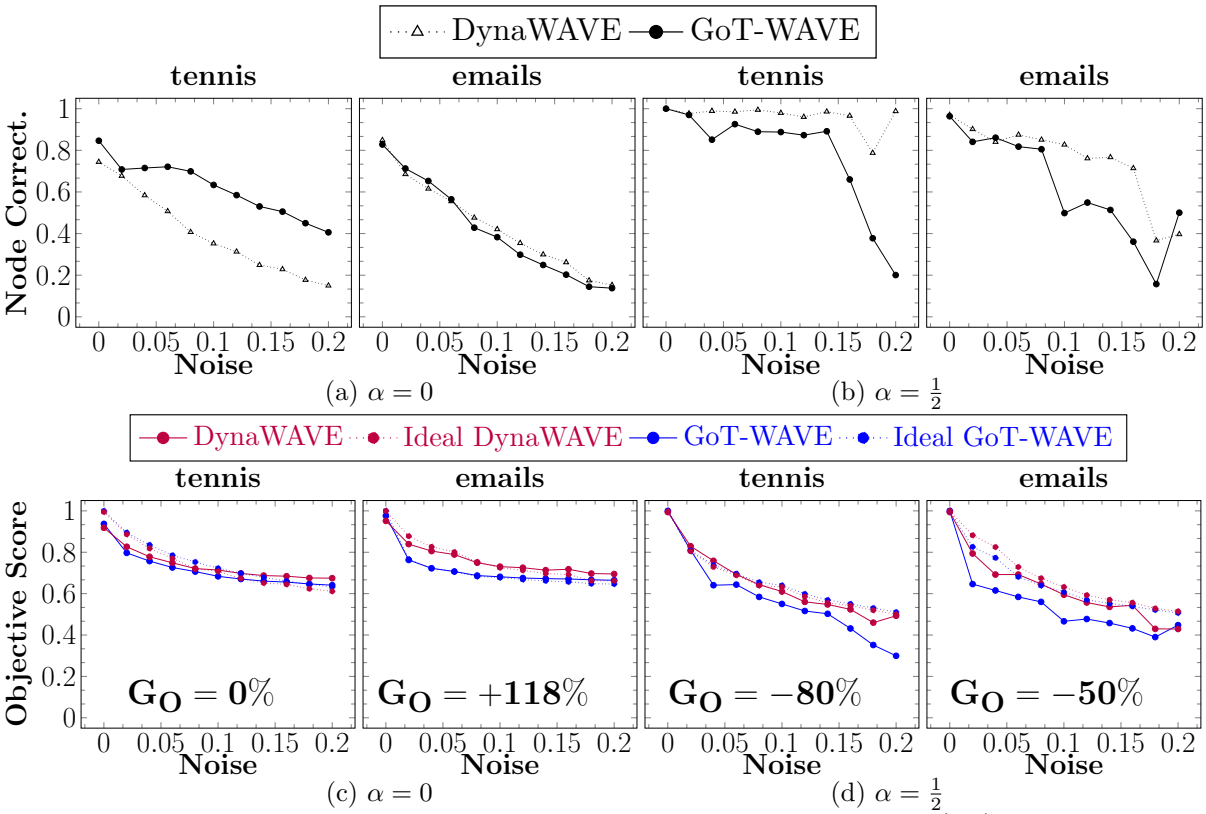


Figure S7: GoT-WAVE against DynaWAVE on directed networks in terms of (a,b) node correctness and (c,d) how well their alignments' objective scores match the scores of ideal alignments. In (a,b), higher node correctness is better. In (c,d), G_O is the relative gain of GoT-WAVE over DynaWAVE.

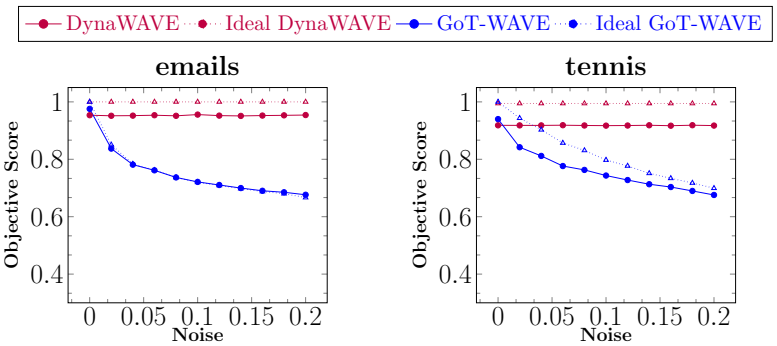


Figure S8: Comparison between GoT-WAVE and DynaWAVE on directed networks in terms of how well their alignments' objective scores match the objective scores of ideal alignments, when only node conservation is optimized ($\alpha = 0$). The noisy versions are generated using the pure directed randomization scheme (i.e., time stamps of two events are never swapped, only the edge direction is swapped). We observe, on one hand, as expected, that DynaWAVE does not distinguish between the original and its randomized versions because DGDVs do not take edge direction into account. On the other hand, GoT-WAVE clearly distinguishes between the original and its randomized versions because GoTs take edge direction into account.

Funding

This material is based upon work partially funded by FCT (Portuguese Foundation for Science and Technology) within project UID/EEA/50014/2013 and by the United States AFOSR (Air Force Office of Scientific Research) [YIP FA9550-16-1-0147]. David Aparício is supported by a FCT/MAP-i PhD research grant [PD/BD/105801/2014].

References

[1] Aparício, D., Ribeiro, P., and Silva, F. A subgraph-based ranking system for professional tennis players. In *Complex Networks VII*. Springer, 2016, pp. 159–171.

[2] Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *Science 286*, 5439 (1999), 509–512.

[3] Erdős, P., and Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci 5*, 1 (1960), 17–60.

[4] Faisal, F. E., and Milenković, T. Dynamic networks reveal key players in aging. *Bioinformatics 30*, 12 (2014), 1721–1729.

[5] Gemmetto, V., Barrat, A., and Cattuto, C. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC infectious diseases 14*, 1 (2014), 695.

[6] Holme, P. Modern temporal network theory: a colloquium. *The European Physical Journal B 88*, 9 (2015), 234.

[7] Hu, H., and Wang, X. Evolution of a large online social network. *Physics Letters A 373*, 12 (2009), 1105–1110.

[8] Hulovatyy, Y., Chen, H., and Milenković, T. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinformatics 31*, 12 (2015), i171–i180.

[9] Isella, L., Stehlé, J., Barrat, A., Cattuto, C., Pinton, J.-F., and Van den Broeck, W. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of theoretical biology 271*, 1 (2011), 166–180.

[10] Kashani, Z. R. M., Ahrabian, H., Elahi, E., Nowzari-Dalini, A., Ansari, E. S., Asadi, S., Mohammadi, S., Schreiber, F., and Masoudi-Nejad, A. Kavosh: a new algorithm for finding network motifs. *BMC bioinformatics 10*, 1 (2009), 318.

[11] Khakabimamaghani, S., Sharafuddin, I., Dichter, N., Koch, I., and Masoudi-Nejad, A. Quatexelero: an accelerated exact network motif detection algorithm. *PloS one 8*, 7 (2013), e68073.

[12] Leskovec, J., Backstrom, L., Kumar, R., and Tomkins, A. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 462–470.

[13] Leskovec, J., Kleinberg, J., and Faloutsos, C. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*, 1 (2007), 2.

[14] Melancon, G. Just how dense are dense graphs in the real world?: a methodological note. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization* (2006), ACM, pp. 1–7.

[15] Michalski, R., Palus, S., and Kazienko, P. Matching organizational structure and social network extracted from email communication. In *International Conference on Business Information Systems* (2011), Springer, pp. 197–206.

[16] Paredes, P., and Ribeiro, P. Towards a faster network-centric subgraph census. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (2013), ACM, pp. 264–271.

[17] Pinar, A., Seshadhri, C., and Vishal, V. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web* (2017), International World Wide Web Conferences Steering Committee, pp. 1431–1440.

[18] Pržulj, N., Kuchaiev, O., Stevanovic, A., and Hayes, W. Geometric evolutionary dynamics of protein interaction networks. In *Pacific Symposium on Biocomputing* (2010), vol. 2009, pp. 178–189.

[19] Ribeiro, P., and Silva, F. G-tries: a data structure for storing and finding subgraphs. *Data Mining and Knowledge Discovery 28*, 2 (2014), 337–377.

[20] Rossi, R. A., and Zhou, R. Leveraging multiple gpus and cpus for graphlet counting in large networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (2016), ACM, pp. 1783–1792.

[21] Rubenstein, D. I., Sundaresan, S. R., Fischhoff, I. R., Tantipathananandh, C., and Berger-Wolf, T. Y. Similar but different: dynamic social network analysis highlights fundamental differences between the fission-fusion societies of two equid species, the onager and grevy's zebra. *PloS one 10*, 10 (2015), e0138645.

[22] Vázquez, A., Flammini, A., Maritan, A., and Vespignani, A. Modeling of protein interaction networks. *Complexus 1*, 1 (2003), 38–44.

[23] Vijayan, V., Critchlow, D., and Milenković, T. Alignment of dynamic networks. *Bioinformatics 33*, 14 (2017), i180–i189.

[24] Vijayan, V., and Milenković, T. Aligning dynamic networks with dynawave. *Bioinformatics* (2017), btx841.

[25] Watts, D. J., and Strogatz, S. H. Collective dynamics of small-world networks. *Nature 393*, 6684 (1998), 440–442.

[26] Wernicke, S., and Rasche, F. Fanmod: a tool for fast network motif detection. *Bioinformatics 22*, 9 (2006), 1152–1153.